Attorney's Docket No.: 15609-019001

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: COMPUTER-IMPLEMENTED METHOD AND SYSTEM TO SUPPORT IN DEVELOPING A PROCESS SPECIFICATION FOR A COLLABORATIVE PROCESS

APPLICANT: FRANK MICHAEL KRAFT

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 342626068 US

August 1, 2003

Date of Deposit

Computer-Implemented Method and System to Support in Developing a Process Specification for a Collaborative Proc ss

FIELD OF THE INVENTION

The present invention generally relates to the development of a specification for a collaborative process involving computer-based participant systems collaborating through exchange of messages over an asynchronous messaging network. More particularly, the present invention is concerned with developing a complete and consistent process specification.

BACKGROUND OF THE INVENTION

Asynchronous messaging is a widely used means of communication between loosely coupled participant systems engaged in a collaborative process such as, e.g., an electronic business process involving distributed business partners. Generally, a collaborative process is one that is characterized by a plurality (two or more) of participants collaborating through exchange of messages in order to accomplish an overall goal or result. Collaborative business processes can be conducted among participants within a single enterprise. They can also involve business partners from different enterprises. Electronic financial controlling/budgeting systems, electronic material resource planning/controlling systems, and electronic purchase/sales order systems are but examples of modern electronic business/commerce applications that may involve collaborative processes of distributed participants, or clients.

Generally, a participant system engaged in a computer-implemented collaborative process can be viewed as comprising the entirety of hardware and software components required to make up a full-fledged participant to the process. In a more specific perspective, a participant system is formed by a software system

implemented and executed on a computer and providing one or more software applications capable of consuming and producing messages received from, and sent to, other participant systems. The participant system may include, or have access to, one or more databases allowing storage of various application-related data such as, e.g., financial figures, fabrication figures, stock figures, etc. The messages exchanged between the participant systems may, e.g., contain requests or reports specifying business actions. The exchange of the messages is asynchronous in that an application sending a message does not need to wait for a remote application to receive that message. Thus, there is no need for all elements of the infrastructure linking the participant systems to be available at all times.

A conventional way of specifying a collaborative process is using UML (Unified Modeling Language) state chart diagrams, one for each participant system. These state chart diagrams specify local states that can be assumed by the participant systems in the course of the collaborative process. Additionally, the state chart diagrams specify local state transitions the participant systems may undergo. Local state transitions are defined by a starting local state, a target local state, a triggering event and a resulting event. The starting local state and the target local state can be different, or be the same. In the first case, the respective participant system experiences a transition from one local state to another, in the second case, a transition-to-self. A triggering event can, e.g., be a message transmitted over the messaging network from another participant system. It can also be a local event unrelated to communication over the messaging network, e.g., a data input through a local user interface or by another local software application. A triggering event can even be a message that is outside the scope of the process analysis. From the sight of the process model, such external message can be triggered as unmotivated as any local event due to, e.g., user action or batch transaction. Similarly, a resulting event can include the sending of a message to another participant system and/or an event local to the respective participant system, e.g., an output of data for display on a monitor or storage in a database.

Optimally, a collaboration specification deals with every possible scenario of occurring messages and local states of the participant system. However, when developing a collaborative process it may easily happen that one or more potentially occurring scenarios are not considered and remain unspecified. If the collaboration process is implemented using a specification that is incomplete, such unspecified situations lead to errors in the collaboration. The user typically does not have a straight solution for this problem. This requires that a provider of customer support be informed, causing costs for maintenance. In some instances these costs can be small compared with losses in revenue or business that may result from the failure.

For humans it is a hard to achieve a fully consistent and complete collaboration specification. Only very experienced personnel will be capable of doing so. This is particularly true as the complexity of the collaboration specification grows dramatically with the number of messages that can be communicated. Even harder is it to achieve this goal of completeness of the process specification in a distributed team of developers, a case normal in today's world of software development. It is therefore highly desirable to provide a developer engaged in developing a specification for a collaborative process with a tool that supports him in achieving a complete and consistent process specification.

SUMMARY OF THE INVENTION

According to one aspect, the present invention provides a computer-implemented method to support in developing a process specification for a collaborative process involving distributed computer-based participant systems exchanging messages through an asynchronous messaging network, the method embodied by a computer program product executable by a computer system and causing, when executed, the computer system to carry out the steps of retrieving from a first storage location, information on local states and local state transitions in relation to

each participant system, this information specifying in relation to each local state transition, starting and target local states of the corresponding participant system and events triggering, and resulting from, the respective local state transition; processing the information retrieved from the first storage location to generate, and store in a second storage location, information on collaboration states and collaboration state transitions of the process, the collaboration states defined by a local state for each participant system and a communication status of each message exchangeable between the participant systems, the collaboration state transitions determined by applying the local state transitions to the collaboration states; upon generation and storage of the information on the collaboration states and collaboration state transitions, retrieving that information from the second storage location; processing the information retrieved from the second storage location to generate information on incompletely specified terminal collaboration states among the collaboration states, an incompletely specified terminal collaboration state being a terminal collaboration state in which at least one message is underway between the participant systems; and generating a result data object containing information on every incompletely specified terminal collaboration state found.

The method of the present invention detects any incompleteness present in the process specification, thus allowing a developer, who may be the single designer of the collaborative process, but may also be part of a distributed team, to eliminate inconsistencies in, and perfect, the process specification early in the design phase. This avoids errors that may occur in practice, which may entail considerable cost and time for rectification. The invention allows for a completion of test cases relevant to the process to be designed. More test cases can be considered and therefore less errors occur at the customer.

The result data object can be stored in a third storage location. The first, second, and third storage locations as used herein refer to any of a database, a file, a

register or set of registers, a memory, a cache, etc that allow to permanently or temporarily store information.

The result data object can be forwarded to a graphical output device to visually present on a display a presentation object indicating every incompletely specified terminal collaboration state found. Presenting the presentation object may include presenting a result list enumerating every collaboration state. The result list may contain in relation to each collaboration state one or more annotation objects indicating, e.g., whether or not the respective collaboration state is an incompletely specified terminal collaboration state.

In one embodiment, the communication status is preferably a binary status indicating whether or not the respective message is being communicated, i.e., is existent.

The step of processing the information retrieved from the first storage location may include generating, and storing in the second storage location, information on a set of virtual global states, the virtual global states being defined each by a local state for each participant system and a communication status of each message, the set of virtual global states comprising states of any combination of local states of the participant systems and communication statuses of the messages.

In a practical implementation, the virtual global states can be represented each by a global state vector composed of first global state vector elements indicating a local state for each participant system and one or more second global state vector elements, one in relation to each message, each second global state vector element indicating a communication status of the respective message, the set of virtual global states comprising states of any combination of values of the first and second global state vector elements.

Processing the information retrieved from the first storage location may further include identifying an initial global state among the virtual global states, this initial global state being one in which at least one local state transition as specified by the information retrieved from the first storage location and involving a local trigger is applicable to the initial global state and no message is underway between the participant systems, said local state transition causing a global state transition from the initial global state to another virtual global state; determining every virtual global state reachable when starting from the initial global state; and determining the initial global state and every virtual global state reachable from the initial global state to be collaboration states.

In another aspect, the present invention provides a computer system to support in developing a process specification for a collaborative process involving distributed computer-based participant systems exchanging messages through an asynchronous messaging network, the computer system provided with a computer program product that, when executed, causes the computer system to carry out the steps of: retrieving from a first storage location, information on local states and local state transitions in relation to each participant system, the information specifying in relation to each local state transition, starting and target local states of the corresponding participant system and events triggering, and resulting from, the respective local state transition; processing the information retrieved from the first storage location to generate, and store in a second storage location, information on collaboration states and collaboration state transitions of the process, the collaboration states defined by a local state for each participant system and a communication status of each message exchangeable between the participant systems, the collaboration state transitions determined by applying the local state transitions to the collaboration states; upon generation and storage of the information on the collaboration states and collaboration state transitions, retrieving that information from the second storage location; processing the information retrieved from the second storage location to generate information on incompletely specified terminal collaboration states among the collaboration

states, an incompletely specified terminal collaboration state being a terminal collaboration state in which at least one message is underway between the participant systems; and generating a result data object containing information on every incompletely specified terminal collaboration state found.

According to yet another aspect, the present invention provides a computer program product providing computer-executable instructions that, when executed by a computer system, cause the computer system to carry out the steps of: retrieving from a first storage location, information on local states and local state transitions in relation to each participant system, the information specifying in relation to each local state transition, starting and target local states of the corresponding participant system and events triggering, and resulting from, the respective local state transition; processing the information retrieved from the first storage location to generate, and store in a second storage location, information on collaboration states and collaboration state transitions of the process, the collaboration states defined by a local state for each participant system and a communication status of each message exchangeable between the participant systems, the collaboration state transitions determined by applying the local state transitions to the collaboration states; upon generation and storage of the information on the collaboration states and collaboration state transitions, retrieving that information from the second storage location; processing the information retrieved from the second storage location to generate information on incompletely specified terminal collaboration states among the collaboration states, an incompletely specified terminal collaboration state being a terminal collaboration state in which at least one message is underway between the participant systems; and generating a result data object containing information on every incompletely specified terminal collaboration state found.

BRIEF DESCRIPTION OF THE DRAWINGS

Hereinafter, the present invention is described in more detail in conjunction with the accompanying drawings in which:

Fig. 1 is a schematic diagram illustrating a system for conducting a collaborative process between distributed participant systems;

Fig. 2 depicts an exemplary activity diagram of the participant systems of Fig. 1;

Fig. 3 schematically illustrates local state chart diagrams related to the activities depicted in Fig. 2;

Fig. 4 is a schematic diagram of a computer system used for carrying out a collaboration checking procedure according to the present invention;

Fig. 5 is a flow chart diagram of steps involved in the collaboration checking procedure;

Fig. 6 illustrates an example of a virtual global state machine related to the activities depicted in Fig. 2;

Fig. 7 illustrates an example of a collaboration state machine model related to the activities depicted in Fig. 2; and

Fig. 8 is an exemplary result object presented as a result of the collaboration checking procedure.

DETAILED DESCRIPTION OF THE DRAWINGS

In Fig. 1, distributed participant systems 10_1, 10_2 (in the following simply referred to as participants) are linked to each other through an asynchronous messaging network generally designated 12. Participants 10_1, 10_2 collaborate to carry out an

electronic process such as, e.g., a trade or other business process. To effect collaboration, participants 10_1, 10_2 exchange messages between each other. Messaging network 12 ensures transport and delivery of such messages in an asynchronous fashion. The messages are produced and consumed by software applications embodying the participants. An order posting system for electronically posting purchase orders for goods with a supplier of such goods, and an order management system electronically handling incoming purchase orders on the part of that supplier are but two examples of software applications that can collaborate through exchange of messages.

The number of participants collaborating via messaging network 12 is at least two but can be any number greater than that. As an example, two additional participants depicted in dashed lines in Fig. 1 may partake in the collaborative process. It is to be understood that the present invention is not intended to be limited to a certain number of participants or a certain minimum or maximum number of participants.

Messaging network 12 may use for message transfer any form of wired or wireless, dedicated or public communications network such as, e.g., a local area network (LAN), a wide area network (WAN), a mobile communications network, or the Internet. Suitable transmission protocols, mechanisms and data formats to effect asynchronous communication between participants 10_1, 10_2 are widely known to those skilled in the art and need not be further detailed.

In the following, a purely illustrative and non-limiting example of a collaborative process is described for sake of facilitating understanding of the invention. This process is a bilateral process carried out between participants 10_1 and 10_2. The activity diagram depicted in Fig. 2 illustrates the way participants 10_1 and 10_2 interact in this process

According to the activity diagram of Fig. 2, participant 10_1, in response to a local triggering event not further specified, produces and emits a message Mx while in a local state S1. Message Mx is delivered through messaging system 12 to participant 10_2. After having sent message Mx, participant 10_1 is in a local state S2. That is, the sending of message Mx is accompanied by a state transition of participant 10_1 from S1 to S2.

Participant 10_2 receives and consumes message Mx in a local state S3. In response to receipt of message Mx, participant 10_2 generates and emits a message My, which is sent over messaging system 12 to participant 10_1. Having sent message My, participant 10_2 remains in local state S3. That is, receipt of message Mx triggers a local state transition of participant 10_2 that is a transition-to-self. Resulting event of this local state transition is the sending of message My.

Participant 10_1 receives message My in local state S2 and remains in this local state even after consumption of message My. Receipt of message My thus triggers a local transition-to-self of participant 10_1, with local state S2 being the starting state and the target state of this state transition. Consumption of message My will cause some form of a local event to occur at participant 10_1. As the type and nature of this local event are of no relevance for the purpose of the present invention, it is not further specified in the activity diagram of Fig. 2.

While in local state S3, participant 10_2 may also produce a message Mz in response to a local triggering event. Message Mz is then sent to participant 10_1. Emission of message Mz is accompanied by a local state transition of participant 10_2 from S3 to S3, i.e., a transition-to-self. Message Mz can be received and consumed by participant 10_1 while the latter is in local state S1. Reception of message Mz triggers a local transition-to-self of participant 10_1 with S1 as the starting state and the target state of this state transition. Again, local events serving as a trigger for the emission of message Mz by participant 10_2 and

resulting from consumption of message Mz on the part of participant 10_1 are not detailed in the activity diagram of Fig. 2.

Fig. 3 illustrates by way of UML state chart diagrams the activities of participants 10_1, 10_2 as shown in Figs. 2. On the left hand side of Fig. 3 a local state chart diagram for participant 10_1 is shown, and on the right hand side of Fig. 3 a similar diagram is shown for participant 10_2. In these state chart diagrams, boxes represent local states assumed by the participants in the course of the process, and arrows represent state transitions. An arrow may point from one box to another; in this case the local state of the corresponding participant system changes as a result of the state transition. An arrow may also loop back to the same box; in this case the local state remains unchanged, that is the corresponding participant system undergoes a transition-to-self. Descriptions associated with the arrows indicate triggering and resulting events of the local state transitions. The first portion of each arrow description specifies a triggering event, while the second portion indicates an event resulting from the state transition..# stands for an event that is local to the corresponding participant system and unrelated to communication activities between participant systems 10_1, 10_2. A local triggering event can, e.g., be an input of data related to ordering, budgeting, accounting or other business procedures by a user. A local resulting event can, e.g., involve the output of such data for storage in a local database, for display on a local monitor, or for printing on a printer or plotter.

UML state chart diagrams such as shown in Fig. 3 provide a simple means of specifying the choreography of the collaboration between various distributed participants. Yet they only specify scenarios that have been explicitly considered by the designer or the team of designers of the collaborative process. In case of collaborative processes that involve a great many of messages and local states of the participants, it is not unlikely for the process designer(s) to inadvertently leave some scenarios unconsidered that nevertheless may occur in practice. Such

unspecified scenarios, in the event of their occurring, will lead to an error and interrupt proper execution of the process.

The present invention provides a tool that enables a process designer to verify completeness and consistency of a collaborative process specification during the design phase. A main feature of this tool is an examination procedure that draws on data that describes the participants' local state transition behavior, and calculates from this data a global state machine model of the collaboration process taking into account both the local states of all participants and the messages that may be exchanged between the participants. Such data can be readily available from UML state chart diagrams as conventionally used by software developers as a means for specifying the process. Once established, the state machine model of the collaborative process allows easy identification of any unspecified scenarios, giving the process designer(s) an early opportunity in the design phase to perfect the process specification.

The task of designing a collaborative process can be assigned to a single developer. In practice, however, such a task is often times assigned to a team of distributed developers. Fig. 4 illustrates an exemplary computer system that can serve as a development environment for the development of software for a computer-implemented collaborative process. The computer system, generally designated 100, comprises at least one computer workstation 110, preferably a number of computer workstations 110, 111, 112 ... coupled to each other via a network 120. In the case of distributed developers partaking in the process design, each developer can work from one of computer workstations 110, 111, 112 ... . Computer workstation 110 comprises a processor 130, a memory 140, a bus 150, and one or more input devices 160 and output devices 170 acting as a user interface. The components of workstation 110 interoperate in a manner conventionally known in the art. A collaboration checking procedure according to the present invention is embodied in a computer program product (CPP) stored in

memory 140 and/or in a storage location outside computer workstation 110, e.g., on a separate program carrier 180.

Computer workstations 110, 111, 112 ... are also referred to as remote computers. They can be servers, routers, peer devices, or other common network nodes, and will typically include many or all of the components indicated in regard of computer workstation 110. Hence, components 130 – 180 of computer workstation 110 may collectively illustrate corresponding components in other computer workstations connected to network 120.

Computer workstation 110 can, e.g., be a personal computer (desktop or notebook computer), a minicomputer, a multiprocessor computer, a mainframe computer, a mobile computing device, a programmable personal digital assistant, or the like. Processor 130 can, for example, be a central processing unit (CPU), a digital signal processor (DSP), a micro-controller, or the like.

Memory 140 symbolizes components that allow to store data and/or instructions permanently or temporarily. Although memory 140 is illustrated in Fig. 4 as a distinct component part of computer workstation 110, it can be implemented by suitable storage resources within processor 130 itself (register, cache, etc.), in nodes of network 120, in other computer workstations, or elsewhere. Specifically, memory 140 can include a read only memory (ROM) portion, e.g., for permanently storing program files, and/or a random access memory (RAM) portion for storing data such as variables and operation parameters. It can be physically implemented by any type of volatile and non-volatile, programmable and non-programmable, magnetic, optical, semiconductor and other information storage means conventionally known in the art, for example, hard disk, floppy disk, CD-ROM, DVD, DRAM, SRAM, EPROM, EEPROM, memory stick, etc.

Memory 140 can store software support modules such as, for example, a basic input/output system (BIOS), an operating system, a program library, a compiler, an

interpreter, communication programs, driver, protocol converters, and application software programs (e.g., text processor, browser, database application, etc.).

The CPP provides in a computer readable manner, program code executable by processor 130. When loaded, the program code causes processor 130 to carry out steps of the before-mentioned collaboration checking procedure. A detailed description of the collaboration checking procedure will be given further below. The CPP controls operation of computer workstation 110 and, if necessary, its interaction with other components of computer system 100. It may be available as source code in any suitable programming language, e.g., C++, or as object code in a compiled presentation. Although Fig. 4 illustrates the CPP as stored in memory 140, it may as well be located on program carrier 180 outside computer workstation 110. If the CPP is stored on program carrier 180, input device 160 will be adapted to allow insertion of program carrier 180 into input device 160 to retrieve the CPP. Rather than storing the program code in memory 140 or on program carrier 180, the CPP can also be provided to processor 130 in the form of program signals delivered through network 120 from a remote computer. The steps embodied by the program code of the CPP can be carried out solely within computer workstation 110 or in a distributed manner by more than one of computer workstations 110, 111, 112 ....

Input device 160 serves to input data and/or instructions for being processed by computer workstation 110. The term input device encompasses, for example, a keyboard, a pointing device (mouse, trackball, cursor direction keys), a reading device for reading information stored on a separate information carrier (e.g., a drive for reading information from a disk, or a card reader for retrieving data from a memory card), a receiver for receiving information transmitted to computer workstation 110 from other components of computer system 100 through a wired or wireless link, a scanner, etc.

Output device 170 serves to present information resulting from processing activities within computer workstation 110. Output device 170 can, e.g., include a monitor or display, a plotter, a printer, or the like. Similarly to input device 160, output device 170, while mainly communicating with a user through visual or other presentation of processing results, may also communicate with other components of computer system 100. Thus, output device 170 may communicate a processing result through a wired or wireless link to a remote recipient.

Input device 160 and output device 170 can be combined in a single device.

Bus 150 and network 120 provide logical and physical connections by conveying instruction and data signals. While connections and communications within computer workstation 110 are handled by bus 150, connections and communications between different computers of computer system 100 and handled by network 120. Network 120 may comprise gateway and router computers dedicatedly programmed to effect data transmission and protocol conversion.

Input and output devices 160, 170 are coupled to computer workstation 110 through bus 150 (as illustrated in Fig. 4) or through network 120 (optionally). While signals inside computer workstation 110 will mostly be electrical signals, signals occurring across network 120 can be any signal type, e.g., electrical, optical, or radio.

Network 120 can be one with wired or wireless access and wired or wireless signal transmission across network 120. It can, e.g., include a LAN, a WAN, a wireless LAN, a public switched telephone network (PSTN), an integrated services digital network (ISDN), or a mobile communications network such as a UMTS (Universal Mobile Telecommunications System), GSM (Global System for Mobile Communication), or CDMA (Code Division Multiple Access) network.

Suitable transmission protocols, mechanisms and data formats to effect communication between computers of computer system 100 can, for example, include Transmission Control Protocol/Internet Protocol (TCP/IP), Hyper Text Transfer Protocol (HTTP), secure HTTP, Wireless Application Protocol (WAP), Unique Resource Locator (URL), Unique Resource Identifier (URI), Hyper Text Markup Language (HTML), Extensible Markup Language (XML), Extensible Hyper Text Markup Language (XHTML), Wireless Application Markup Language (WML), Electronic Data Interchange (EDI), which governs the electronic exchange of business information between or within organizations and their IT (Information Technology) infrastructure in a structured format, Remote Function Call (RFC), an application programming interface (API), etc.

In the following, one embodiment is described of a collaboration checking procedure for conducting an examination of a design for a collaborative process using the computer topology illustrated in Fig. 4. Fig. 5 depicts by way of a flow chart diagram, main steps of the collaboration checking procedure.

First, a user effects start of execution of the program code included in the CPP. To this end, a control signal is generated through user-operation of an activator element. The activator element can, e.g., a control button displayed on a graphical user interface created on a monitor by a software program application resident on computer workstation 110. The user can activate the control button on the screen by means of a mouse pointer. Activation of the control button generates a control signal that causes processor 130 to execute the collaboration checking procedure (step 200 in Fig. 5)

In executing the collaboration checking procedure, processor 130 first retrieves data specifying the process to be examined from a first storage location DB1 (step 210 in Fig. 5). This process specification data includes information on local states and local state transitions of the process participants along with associated triggering and resulting events. In one embodiment, the participants' local states

can be made up of a single state each. In an alternate embodiment, they can be made up of a plurality of states each associated with a respective one of various sub-systems of the respective participant. Such a sub-system can, e.g., be a software object representing a business object used in a business software program application. The process specification data can be stored in a database, a file, or any other organizational structure. The first storage location DB1 can be within memory 140 of computer workstation 110, or on any other computer workstation 111, 112 ... connected to computer system 100, or even on a network server of network 120. The first storage location DB1 can be a shared resource giving all developers in a development team access to the data in the first storage location DB1.

Taking the example process specified by the UML state chart diagrams of Fig. 3, the participants' 10_1, 10_2 process specification data can, e.g., be stored in the first storage location DB1 as a list of local state transition vectors (LSTV) each representing a single local state transition (LST) such as illustrated in the following table:

LSTV	LST
(10_1; S1; S2; #; Mx)	1
(10_1; S2; S2; My; #)	2
(10_1; S1; S1; Mz; #)	3
(10_2; S3; S3; #; Mz)	4
(10_2; S3; S3; Mx; My)	5

The first element in each LSTV vector represents the participant affected by the particular local state transition, i.e., participant 10_1 or 10_2. The second and third vector elements represent the starting local state and the target local state, respectively. The fourth and fifth elements in each LSTV vector represent the triggering and resulting events of the local state transition considered, with the same denotation used for these events as in the state chart diagrams of Fig. 3.

One skilled in the art will be readily able to verify that the five local state transition vectors listed above cover all the scenarios specified by the state chart diagrams of Fig. 3.

Having retrieved the process specification data from first storage location DB1, processor 130, under control of the program code provided by the CPP, processes this data to generate, and store in a second storage location DB2, information representing a set of global state vectors (step 220 in Fig. 5). In one embodiment, each global state vector is composed of a plurality of first global state vector elements, one in relation to each participant, and a plurality of second global state vector elements, one in relation to each message Mx, My, Mz. The first global state vector elements indicate a local state of the respective participant, and the second global state vector elements indicate whether or not the respective message is underway between participants 10_1 and 10_2. To this end, the second global state vector elements can conveniently be binary values.

In an alternate embodiment, the global state vectors are composed of a plurality first global state vector elements, one in relation to each participant, and second global state vector elements in relation to those messages only that are underway between the participants. Thus, a global state vector will have no second global state vector element if no message is underway, and one or more second global state vector elements if a corresponding number of messages is underway. Each second global state vector element will indicate a particular one of messages Mx, My, Mz that is underway.

In any event, the global state vectors are formed in dependence of each participant's local state and each message's communication status, i.e., whether the respective message is being communicated (underway) or not. For the purpose of the example process described herein, it is assumed in the following that the global state vectors contain a binary value for each message.

Using global state vectors an imaginary global state machine can be defined whose states, hereinafter referred to as virtual global states, are each represented by a respective global state vector and whose current virtual global state hence depends on the current local states of all participants of the collaborative process as well as the state of presence or non-presence of each message, i.e., whether or not the corresponding message is being communicated.

The second storage location DB2 can, e.g., be within memory 140 of computer workstation, as is illustrated in Fig. 4. Alternately, it can be a storage location external to computer workstation 110, for example, in a node of network 120 or in another one of computer workstations 111, 112 ... connected to network 120. The global state vectors can be stored at the second storage location DB2 in any organizational form such as a simple data set, a database, a file, etc. The global state vectors can even be kept in one or more internal registers of processor 130 and deleted, or removed to another location, after completion of the calculations performed by processor 130 in the course of the collaboration checking procedure.

The set of global state vectors determined by processor 130 is comprised of vectors of any combination of values of the first and second global state vector elements. The total number N of global state vectors to be generated by processor 130 can be calculated by:

$$N = \prod m_i \times 2^k \qquad (i = 1, 2, \ldots n)$$

where $\prod$ represents a mathematical product, m_i represents the total number of local states of an individual process participant 10_i, with $\prod m_i$ thus representing the product of the total local state numbers of all process participants 10_1, 10_2 ..., and k represents the total number of messages that may occur.

In the example process specified by the UML state chart diagrams of Fig. 3, the total number of local states of participant 10_1 is 2 (S1, S2) , the total number of

local states of participant 10_2 is 1 (S3), and the total number k of messages is 3 (Mx, My, Mz). The total number N of global state vectors, hence virtual global states, for the example process is therefore:

$$N = 2 \times 1 \times 2^3 = 16$$

The global state vectors will have five vector elements each, two indicating the local states of participants 10_1,10_2 and three others indicating presence or non-presence of messages Mx, My, and Mz. The following table illustrates all 16 global state vectors (GSV) and corresponding virtual global states (VGS) that can be formed in the case of the example process:

GSV	VGS
(S2; S3; 1; 1; 1)	0
(S2; S3; 1; 1; 0)	1
(S2; S3; 1; 0; 1)	2
(S2; S3; 1; 0; 0)	3
(S2; S3; 0; 1; 1)	4
(S2; S3; 0; 1; 0)	5
(S2; S3; 0; 0; 1)	6
(S2; S3; 0; 0; 0)	7
(S1; S3; 1; 1; 1)	8
(S1; S3; 1; 1; 0)	9
(S1; S3; 1; 0; 1)	10
(S1; S3; 1; 0; 0)	11
(S1; S3; 0; 1; 1)	12
(S1; S3; 0; 1; 0)	13
(S1; S3; 0; 0; 1)	14
(S1; S3; 0; 0; 0)	15

In the above table, the first element of each global state vector indicates the local state of participant 10_1, the second vector element indicates the local state of participant 10_2. The last three vector elements are associated to messages Mx, My, and Mz, in that order. Here, a binary value 1 may symbolize presence of a particular message, whereas a value 0 may represent non-presence of that message, or conversely.

As an example, virtual global state VGS 14 defines a constellation where participant 10_1 is in local state S1, participant 10_2 assumes local state S3, message Mx is non-present (assuming an interpretation of "0" as being indicative of non-presence), message My is also non-present, and message Mz is underway.

Having generated and stored information representing the set of global state vectors in the above-described manner, processor 130 proceeds by generating global state transition information that describes the global state transition behavior of the above-mentioned virtual global state machine having all the virtual global states. Specifically, processor 130 determines all global state transitions that may occur in the virtual global state machine as a consequence of the local state transitions as specified by the process specification data stored in database DB1.

Even more specifically, taking one of the local state transitions and one of the virtual global states, processor 130 checks if the starting local state of that local state transition is identical to the local state indicated for the corresponding participant in the virtual global state. If the starting local state of the local state transition is different from that participant's local state as indicated in the virtual global state, the local state transition is impossible to occur in that virtual global state, i.e., it is not applicable. However, if there is identity between the starting local state of the local state transition and the local state indicated for the participant in the virtual global state, it is additionally checked if the trigger of the local state transition is a local event or an incoming message that is indicated as

present, or underway, in the virtual global state. If this is true, the local state transition may occur in the virtual global state, i.e., it is applicable. On the other hand, if the trigger of the local state transition is a message that is non-present in the virtual global state considered, the local state transition is impossible to occur in that virtual global state, i.e., it is not applicable.

Once a local state transition has been found applicable to a virtual global state, a new, or target, virtual global state is determined that will result from that local state transition. To this end, it is determined what is the target local state of the participant experiencing the local state transition. Taking into account the fact that any triggering message is consumed as a consequence of the local state transition and therefore ceases to exist, and taking also into account any resulting message that may come into existence as an accompaniment to the local state transition, the new virtual global state can be easily determined based on the target local state, which is taken as the local state of the corresponding participant in the new virtual global state, and also based on the consumption and production of any messages.

To facilitate understanding of the above, a description of an example follows where it is checked which of local state transitions LST 1 – LST 5 of the example process are applicable to virtual global state VGS 3 from the table above. One finds easily that only two local state transitions are applicable, while the remaining three are impossible to occur in VGS 3. The applicable local state transitions are LST 4, defined by vector (10_2; S3; S3; #; Mz), and LST 5, defined by vector (10_2; S3; S3; Mx; My). LST 4 is a local state transition of participant 10_2 from local state S3 to that same state, i.e., a transition-to-self, in response to a local triggering event #, with the emission of message Mz as the resulting event. As the starting local state of LST 4 is identical to the local state indicated for participant 10_2 in virtual global state VGS 3, namely S3, and as the trigger of the local state transition is an event local to participant 10_2, the conclusion is that LST 4 is applicable to VGS 3.

Application of LST 4 to VGS 3 results in a global state transition from VGS 3 to VGS 2. No message is consumed in this local state transition as it is triggered by a local event. On the other hand, LST 4 causes production and emission of message Mz. From a global perspective, LST 4 thus adds a further message, Mz, to message Mx already existent in VGS 3. Target virtual global state of the global state transition caused by LST 4 is therefore a virtual global state in which participant 10_1 is in local state S2 (participant 10_1 is left unaffected by this local state transition), participant 10_2 is in local state S3 (participant 10_2 experiences a transition-to-self), and both messages Mx and Mz are existent. The one virtual global state meeting this scenario is VGS 2.

Local state transition LST 5 is likewise applicable to virtual global state VGS 3. LST 5 is a transition-to-self of participant 10_2 from local state S3 to that same state, with message Mx triggering the transition and message My being emitted as a result. Therefore, LST 5 has a starting local state identical to that which is required by VGS 3 as the local state of participant 10_2, S3. Moreover, message Mx, the trigger of LST 5, is a message existent in VGS 3, as indicated by the value "1" of the third vector element in the global state vector corresponding to VGS 3. Consequently, local state transition LST 5 is possible to occur in virtual global state VGS 3.

Application of LST 5 to VGS 3 results in a global state transition from VGS 3 to VGS 5. As before, participant 10_1 remains unaffected by this local state transition, and participant 10_2 undergoes a transition-to-self. However, LST 5 leads to consumption of message Mx while message My is produced by participant 10_2. Therefore, target virtual global state of the global state transition caused by LST 5 is a virtual global state in which participants 10_1, 10_2 are in local states S2 and S3, respectively, and message My is present only. Virtual global state VGS 5 represents this scenario.

The remaining local state transitions LST 1, LST 2, and LST 3 are not applicable to virtual global state VGS 3. LST 1, represented by vector (10_1; S1; S2; #; Mx), requires local state S1 to be the starting local state of participant 10_1, much like LST 3, represented by vector (10_1; S1; S1; Mz; #). This, however, contrasts with the requirement of VGS 3 that participant 10_1 be in local state S2. Finally, LST 2, represented by vector (10_1; S2; S2; My; #), needs message My as the trigger. In virtual global state VGS 3, however, no message My is underway as indicated by the value "0" of the fourth vector element in the global state vector corresponding to VGS 3. Therefore, local state transitions LST 1, LST 2, and LST 3 are impossible to occur in virtual global state VGS 3.

Generally, processor 130 checks applicability of every local state transition in relation to every virtual global state, and then determines for each applicable situation what is the target virtual global state that will result from application of a particular local state transition to a particular virtual global state. In this way, processor 130 obtains global state transition information that is added to storage location DB2 (step 230 in Fig. 5). Thereafter, storage location DB2 includes all the information necessary to create a global state transition graph of a virtual global state machine having all the virtual global states. The virtual global states form nodes of the graph, and connections between nodes represent global state transitions experienced by the virtual global state machine when subjected to local state transitions. As with the local state transitions, global state transitions may involve a transition from one virtual global state to another, or a transition-to-self looping back to the same virtual global state.

Fig. 6 illustrates by way example a global state transition graph established for the example process specified by the UML state chart diagrams of Fig. 3. The nodes of this graph represent the virtual global states VGS 0 – VGS 15, and arrows connecting the nodes represent all the global state transitions that result from applying the local state transitions LST 1 – LST 5 to the virtual global states VGS 0 – VGS 15 in the manner described above. One ordinarily skilled in the art will be

readily able to verify the global state transitions depicted in the graph of Fig. 6. In particular, one can easily recognize in the graph of Fig. 6 two arrows that represent the two exemplary global state transitions discussed above in connection with virtual global state VGS 3, namely an arrow pointing from VGS 3 to VGS 5 and another leading from VGS 3 to VGS 2.

Having entirely generated, and stored in storage location DB2, all the information on the virtual global states and the global state transitions in the above manner, processor 130 uses this information to determine a subset of valid global states from the virtual global states (step 240 in Fig. 5). That is, not all virtual global states may actually occur when carrying out the collaborative process under examination. In practice, the collaborative process will start with no message being underway. Further, there must be specified at least one local state transition applicable to one of the participants to the collaborative process at the time the collaborative process starts. The trigger of such local state transition must be an event local to the corresponding participant, since no message is underway at the time the collaborative process starts. These restraints will limit the number of virtual global states that may actually be entered when carrying out the collaborative process to a subset of virtual global states called valid global states hereinafter.

To determine the valid global states, processor 130 retrieves the information from storage location DB2 and proceeds with identifying one or more initial global states among the virtual global states, with an initial global state being defined as one in which at least one local state transition as specified by the local state transition data and involving a local trigger is applicable to the initial global state and no message is being communicated between the participants. Further, the local state transition must not result in a global state transition looping back to the same starting global state.

Referring to the example process specified by the state chart diagrams of Fig. 3, the only virtual global states that have no messages underway are VGS 7 and VGS 15. In VGS 7, participant 10_1 is in local state S2 and participant 10_2 is in local state S3. A comparison of local state transitions LST 1 – LST 5 shows that LST 4 is applicable to VGS 7 since it has a local trigger and requires a starting local state that is identical to the local state of one of participants 10_1, 10_2 in VGS 7. Application of LST 4 to VGS 7 leads to VGS 6, which is different from the starting virtual global state VGS 7. Therefore, virtual global state VGS 7 is an initial global state. No other local state transition is applicable to VGS 7 though.

As for VGS 15, this is a virtual global state to which local state transitions LST 1 and LST 4 are applicable, both requiring a local trigger and having starting local states identical to the local state of one of participants 10_1, 10_2 in VGS 15. Application of LST 1 and LST 4 to VGS 15 leads to VGS 3 and VGS 14, respectively. Again, the target virtual global states are different from the starting virtual global state. Hence, virtual global state VGS 15 is another initial global state.

Next, processor 130 determines for each initial global state found, all virtual global states that can be reached by the virtual global state machine when starting from an initial global state. The initial global state and every virtual global state reachable from that initial global state are then determined to be valid global states.

As an example, by screening the global state transition graph of Fig. 6 for valid virtual global states, or nodes, one can easily find that, in addition to starting nodes 7 and 15, nodes 2, 3, 4, 5, 6, and 14 are valid nodes. These are the only nodes that can be reached in the global state transition graph of Fig. 6 when starting from either of nodes 7 and 15. The remaining nodes 1, 8, 9, 10, 11, 12, and 13, cannot be reached from nodes 7 and 15 and therefore represent invalid virtual global states. Eliminating in the global state transition graph of Fig. 6 any invalid nodes

and any connections between invalid nodes and between invalid and valid nodes results in the partial graph shown in Fig. 7, which represents a valid global state machine model of the collaboration process specified by the state chart diagrams of Fig. 3.

In an alternate embodiment, only one virtual global state may be allowed as an initial global state. Further, the definition of an initial global state may be altered or supplemented to refer to a virtual global state for which no state transition exists of which it is the target virtual global state.

Hereinafter, a valid global state machine model of a collaborative process is referred to as a collaboration state machine model of the process. Its states are called collaboration states and are constituted by the valid virtual global states forming the valid nodes of the global state transition graph. A collaboration state machine model for a process specification under examination is thus defined by the entirety of valid virtual global states of the process as specified and the entirety of valid global state transitions, i.e., global state transitions involving valid virtual global states both on the starting and the target sides. Such valid global state transitions are referred to hereinafter as collaboration state transitions. In a step 250 in Fig. 5, processor 130 labels every valid global state stored in storage location DB2 to be a collaboration state, by adding a suitable annotation object.

Subsequently, processor 130 analyzes the collaboration state machine model for presence of incompletely specified terminal collaboration states (step 260 in Fig. 5). A terminal, or dead-end, collaboration state is one that is never the starting state of a collaboration state transition that would lead to a different collaboration state. In other words, a terminal collaboration state is a node in the collaboration state machine model that has no outbound state transitions and therefore can not be left.

There may exist one or more terminal collaboration states where no messages are underway. Such terminal collaboration states may be acceptable and are not deemed incompletely specified. Only a terminal collaboration state where one or more messages are underway indicates an incompleteness in the process specification. In order for the process specification to be complete and consistent, any such incompletely specified terminal collaboration states need to be corrected by the process designer(s).

To explain this by way of an example, it is referred to Fig. 7. There, it can be seen that while each of collaboration states 2, 3, 4, 5, 7, 14, and 15 can be left through a collaboration state transition, collaboration state 6 is a terminal (dead-end) state. This collaboration state is associated with three collaboration state transitions, one pointing from collaboration state 4 to collaboration state 6, another one leading from collaboration state 7 to collaboration state 6, and a third one beginning with collaboration state 6 but looping back to its origin, i.e., a transition-to-self. There is no collaboration state transition originating from collaboration state 6 and leading to a different collaboration state.

Terminal collaboration state 6 points at an unspecified scenario in the design of the collaboration process. In fact, collaboration state 6 represents a situation where participant 10_1 is in local state S2, participant 10_2 is in local state S3, and message Mz is on its way. The specification of the example process does not deal with this situation; it leaves unanswered how message Mz shall be processed. The process specification is therefore incomplete and needs to be amended accordingly. On the other hand, if no message were underway in collaboration state 6, the situation might be tolerable as no processing of a message were necessary. In such case, collaboration state 6 would not be deemed incompletely specified.

In a following step designated 270 in Fig. 5, processor 130 effects generation, and preferably storage in a third storage location, of a result data object containing

information on every incompletely specified terminal collaboration state found. The third storage location can be a database, a file, or any other suitable data repository that allows to hold data in an organized or structured manner. The third storage location can be a resource accessible by some or all of the software developers involved in designing the process so as to enable plural developers to learn of the examination result of the collaboration checking procedure. To this end, it is conceivable that the result data object is stored on a network computer (server) of the computer system 100. Of course, storing the result data object in memory 140 of computer workstation 110 or anywhere else in the computer system 100 is also possible. For sake of illustration only, the result data object is shown in Fig. 4 as being stored in memory 140 of computer workstation 110 at a storage location indicated by a reference symbol DB3.

In a preferred embodiment, the result data object can be forwarded to output device 170 to visually present on a display a graphical presentation object indicating every incompletely specified terminal collaboration state found. Specifically, processor 130 may effect display, or printing, of a result list containing at least every incompletely specified dead-end collaboration state. In one embodiment, the result list specifies every virtual global state as determined by processor 130 and indicates for each virtual global state whether or not the respective virtual global state is valid, whether or not it is a dead-end state, and whether or not it represents an incompleteness in the process specification. In an alternate embodiment, the result list may specify only those valid virtual global states that are found to be incompletely specified terminal collaboration states.

Part of an exemplary result list that may be presented to the developer on the graphical user interface displayed on output device 170 of computer workstation 110 is depicted in Fig. 8. There, a number indicating the respective virtual global state is followed by a first annotation object AO1 that indicates whether or not the particular virtual global state is valid or invalid (i.e., a collaboration state). A second annotation object AO2 indicates in relation to every virtual global state whether or

not the respective virtual global state is a dead-end state or a live-end state. A third annotation object AO3, finally, indicates whether or not the process specification under examination is complete in relation to the respective virtual global state.

Summarizing briefly the above collaboration checking procedure, processor 130 retrieves in response to a user-effected control signal, process specification data from a first storage location DB1 and generates, and at least temporarily stores in a second storage location DB2, information on virtual global states, global state transitions, and which of the virtual global states are valid global states, i.e., collaboration states. Thereafter, processor 130 retrieves the information stored in the second storage location DB2 to determine incompletely specified dead-end collaboration states. Finally, processor 130 generates a result data object containing information on every incompletely specified terminal collaboration state found. The result data object can be stored in a third storage location and/or forwarded to a graphical output device for visual presentation of a presentation object on a display, for example, in the form of a result list indicating every incompletely specified dead-end collaboration state.